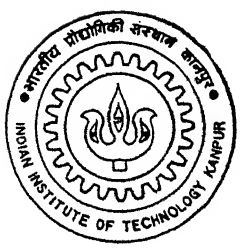
Sandhi-Analysis using Two-Level Rules

by

SANDEEP NANGIA



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

APRIL, 1995

Sandhi Analysis using Two-level Rules

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Master of Technology

by

Sandeep Nangia

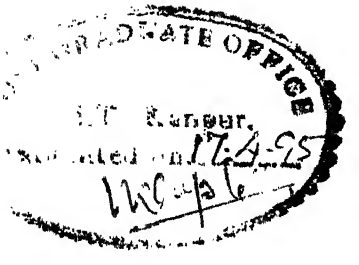
to the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

April 1995

CERTIFICATE

Certified that the work contained in the thesis entitled "Sandhi Analysis using Two-level Rules", by "Sandeep Nangia", has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.



Rajeev Sangal

(Prof. Rajeev Sangal)
Department of Computer Sci. & Engg.,
Indian Institute of Technology,
Kanpur.

April 1995

Abstract

When two sounds come together they are combined for the ease of pronunciation. This phenomenon is called *sandhi*. The problem of analysis of sandhi is a very important one in the building of a Machine Translation (MT) system for Sanskrit. Sandhi-rules in Sanskrit differ depending upon the context where sandhi takes place — i.e. at the sentence level between words, within the formation of compounds or for the formation of derived forms of words and root forms. An attempt is made to build a sandhi-analyzer at sentence level by using finite-state transducers as means to map the final combined form of words (in which sandhi has taken place) to the form in which sandhi has been split. This is done by converting sandhi-rules into *two-level rules* which are then converted into finite-state transducers.

Acknowledgements

The business-like attitude, appreciation (nay, demand) for initiative and innate sharpness are the things that I have really come to admire in Prof. Rajeev Sangal, my thesis supervisor, in this now 21 month old association with him. He has given me a lot to learn.

Dr. Vineet Chaitanya has contributed a lot by his suggestions in our discussions. Dr. D.K. Jha was ever-ready to solve my problems relating Sanskrit grammar. I owe special thanks to him. Other members of Akshara Bharati group too helped in manifold ways.

Vineet "Pyjama" Gupta, Arun "Gemini" Goyal, Vidyut "Buffalo" Verma , Kamlesh "Finger" Barot, Rajababu "Suspended" Kommula and T.A.N. "Gaali" Reddy all have propped me up when I was despondent and have shared my happiness when I was gleeful.

I thank all the inmates of Ramakrishna Mission Ashrama, Kanpur especially Subrato Maharaj for their wonderful company (er ... whenever I would go and meet them !)

My parents and Mataji have supported me all through this endeavour. But, the debts I owe them are far far weightier than I owe anyone else.

Sandeep Nangia

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In building of a machine translation system for any source language to another language, it is required that the words in the source language be analyzable. Amongst the properties that need to be found out for each word presented to a machine translation (MT) system are — part of speech, gender, number, person etc. The module that is able to analyze a given word providing the aforesaid information is called *morphological analyzer.*

In the simplest model for such a module, an exhaustive listing of words in the source language with their associated properties can be used. While this approach is sufficient for toy systems, for the building of MT systems useful in practice, this approach is clearly impractical. For a language like English, it is possible that an exhaustive dictionary be able to handle the words encountered in most of the English sentences[1] But for Indian languages such an approach may yield unwieldy dictionaries. It is thus desirable that the morphological analyzer be constructed using a rules component and a lexicon where the rules component dictates how different words may be derived from baser forms in the lexicon and combined with each other.

[1]Even in English there do occur problems. A pertinent example is compound formation. But the problems are indeed less severe than other more morphologically rich languages.

1.1 Sandhi

A new dimension is added to this problem in the analysis of Sanskrit text. Not only is the process of compound formation and derivation process much more intricate, an additional problem of *sandhi* amongst adjacent words in a sentence is also present. *Sandhi* is the process in which two sounds combine on coming close to each other. Sandhi not only occurs between words in a sentence but also during compound formation (i.e. in the formation of *samasa*). The phenomenon of sandhi, though, is not unique to Sanskrit. In other Indian languages also this phenomenon is present but limited to compound formation or in words directly borrowed from Sanskrit (wawsam words).

Further, in Sanskrit it it mandatory[2] to do sandhi amongst adjacent words in the written text [19, 20, 21]. Though, primers of Sanskrit sometimes do not employ sandhi between adjacent words for pedagogical reasons, in literature sandhi is universally used.

Thus, for a MT system for Sanskrit it is essential to have a module which can break combined words at proper places. An additional constraint that must be used in breaking sandhi is that only those breakings of sandhi are valid which result in a sequence of meaningful words. In particular, each of the words must be morphologically valid. Chapter 2 discusses the problem of sandhi in detail.

This thesis attempts to build a sandhi analyzer for sentences in Sanskrit. We first take a look at the related work done previously in this area.

[2]In spoken language, though, the speaker has a choice between doing or not doing sandhi between adjacent words.

1.2 Related Previous Work

Bhattacharya [6] shows how to build a rudimentary morphological analyzer of Sanskrit. Much more extensive work on building morphological analyzer has been reported by Ramanujan [17]. Akshar Bharati et al. [10] outlines a paradigm-based approach for morphology of Indian languages. Sproat [22] provides an extensive survey of various techniques that have been used for morphological/phonological analysis.

A few techniques that have been described in literature are briefly described below:

Fonol Fonol [7] tries to model the ordered sequence of rules in traditional generative phonology. These rules are of the form $A \rightarrow B/\alpha\_\_\_\_\beta$ which means that the string of symbols A should be replaced by the string of symbols B when preceded by α and followed by β.

Chart Parsing Sproat [22] mentions use of chart parsing for analysis of morphology. The essential idea in this is the application of reverse of phonological rules (called *morphographemic rewriting* and then using a dictionary to lookup the sequence of strings generated.

KIMMO KIMMO has been a very successful model for morphological and phonological analysis. The essential idea here is to model the morphotactics[3] using a finite state machine. The phonological rules are modeled using finite state transducers — i.e. finite state machines accepting a sequence of symbol pairs, one symbol from the lexical component and the other the final forms. This method was proposed originally by Kimmo Koskenniemi [13] and is called KIMMO after his first name. Antworth [1] discusses an implementation of KIMMO on personal computers and calls it PC-KIMMO. The author of this thesis has used this implementation for the work described herein.

[3] i.e. the ordering restrictions of morphemes

While the essential framework comprising KIMMO is the use of finite state transducers for modeling phonology, an alternate rule formalism called *two-level rules* [13, 1] is used which is further compiled into finite state transducers. Kay and Kaplan [12] give the mathematical foundations for finite state computational phonology and morphology. Ritchie [18] describes the languages generated by two-level rules.

Extensions to KIMMO KIMMO has two components: the *rules component* which describes the phonology of the language and the *lexical component* which describes the morphotactics of the language. Bear [8] proposes an additional level – a word grammar based on unification scheme for directed acyclic graphs. This additional level comprises a context-free grammar with unification between feature-structures of the morphemes. Antworth [3] describes another such implementation and gives a tutorial introduction to the use of word-grammar for ensuring morphotactics.

1.3 The Work Done

An attempt is made to solve the sandhi-analysis problem in Sanskrit using PC-KIMMO. Thus for the various sandhi rules equivalent two-level rules were written down and converted to finite-state transducers. The lexical component of PC-KIMMO is used to ensure the restriction that only valid words can participate in sandhi at sentence level in Sanskrit. The lexical component comprised around 34,000 words. Included in these words are all the words found in the Gita.

1.4 Organization of the thesis

Chapter 2 describes the phenomenon of sandhi in Sanskrit. Chapter 3 describes KIMMO. Chapter 4 describes the process of conversion of sandhi-rules into two-level rules. Conclusion and further possible work come in Chapter 5.

Appendix A lists the English characters used for the devanagari script. Appendix B gives a sample rule file of two-level rules. A sample input file for PC-KIMMO has been given in Appendix C. Appendix D gives a sample lexical component file for PC-KIMMO. Appendix E shows an example sample run of PC-KIMMO.

In this thesis, the typewriter font is used for denoting devanagari script. Thus `rAma` would mean *rāma* in devanagari script.

Chapter 2

Sandhi

Panini's Ashtadhyayi is the source book for rules of Sanskrit grammar. Ashtadhyayi comprises of approximately four thousand sutras (aphorisms)[1] divided into eight chapters each of which comprises of four sections. Certain modifications and corrections in Panini's rules were made by Katyayan in his vartikas. Further Patanjali has commented and expanded on these rules in Mahabhashya. These three books viz. ashtadhyayi, vartikas, and the mahabhashya together form the source material on Sanskrit grammar. These books have further inspired many more commentaries and expansions by later grammarians. Later the ashtadhyayi was rearranged (according to the topics in Sanskrit grammar where these rules get applied) in Bhattoji Dikshit's Siddhantakaumudi. About 1400 of the most often used sutras were commented upon by Varadaraja's in his Laghusiddhantakaumudi. Both these books rely upon the derivation process given by Panini. Teaching and learning of Sanskrit in the traditional way uses these two books - Laghusiddhantakaumudi for the introduction and then Siddhantakaumudi for following up.

[1]The notation *Pa. a.b.c* is used to denote the cth sutra in the bth section of the ath chapter in Ashtadhyayi. Thus Pa. 8.2.1 denotes the first sutra in the second section of the 8th chapter of Ashtadhyayi.

The Paninian derivation process and in particular, sandhi is described in this chapter. The texts of Candace J. Simpson [21], Whitney's grammar [24], Buckenell's manual on Sanskrit [9] and Ramakrishnamacharyulu's personal communication [16], S.C.Vasu's Siddhantakaumudi [23] have been used to write this introduction.

2.1 Introduction to Sandhi in Sanskrit

When two sounds in Sanskrit come close to each other they combine for the ease of pronounciation. This combination is known as *sandhi.* The process of sandhi includes the modifications that take place when two words in a sentence come together, the modifications that take place when a prefix is attached to a verb and also the modifications that take place while forming compounds using two words and also the changes that take place during the derivational or inflectional process of forming final word or verb forms. The rules of combination are in some respect different according as they apply to:

1. to the internal make up of the word by the addition of derivation and inflectional endings to roots and stems and when prefixes are joined to verbs. These changes are referred to as *internal sandhi.*

2. to the more external putting together in compounds and the yet more external putting together of words in a sentence. The changes that take place in this context are known as *external sandhi.*

It should be noted that in the process of formation of compounds (samAsa) certain other changes take place besides sandhi. In this thesis an attempt is made to solve the problem of sandhi when words in a sentence are put together. Thus, unless mentioned otherwise, sandhi is used in this thesis as the changes that take place when different words (padas) come together in a Sanskrit sentence. Also, only the aspects of classical Sanskrit are considered. Thus accents and other special rules pertaining to Vedic Sanskrit are ignored.

2.2 Issues Involved

When we deal with the written form, some additional issues arise because the written form does not have intonation information. As a result, certain things which are explicit (and quite unambiguous) in the spoken form become implicit (and possibly ambiguous) in the written form. Sometimes there are more than one conventions for writing. Two issues are of importance here, as described below:

Handling blanks In traditional usage in manuscripts and inscriptions the whole material in a sentence is put together without separating any word from another. But in modern practice unless sandhi occurs we do put one (or more) blank between two succeeding words. Thus the following mantra from Rig Veda (X.125) when written in the modern form would be written like:

```
ahaM ruxreBirvasuBiScarAmyahamAxiwyEruwa viSvaxevEH |
ahaM miwravaruNoBA biBarmyahaminxrAgnI ahamaSvinoBA ||
```

While in a manuscript it would be written in the following fashion:

```
ahaMruxreBirvasuBiScarAmyahamAxiwyEruwaviSva
xevEH | ahaMmiwravaruNoBAbiBarmyahaminxrAgnI
ahamaSvinoBA ||
```

Handling Avagraha Another point is the use of avagraha Z. Avagraha in a written text indicates that an a has been elided (or dropped) at that place. Sometimes a double avagraha is also used to indicate the elision of A. The following example illustrates the use of avagraha.

axyawve + api → axyawveZpi

Even though the usual practice is to put an avagraha in cases where a gets elided some texts do not follow this. Thus the axyawveZpi may be found

written as axyawvepi in another text. Also in the case of *savarna sandhi* (when a like vowel follows a like vowel) this elision of A is usually not marked. This practice is also not universal as sometimes a double avagraha marks it instead.

Another issue that is of importance is the order in which successive words like inxra A ihi are to be combined. The correct order in which they are to be combined is (inxra + A) + ihi to give the intermediate form inxrA + ihi which gives the final form inxreHi and not as inxra + (A + ihi) which would have given the final form inxreZhi.

2.3 Sandhi in Ashtadhyayi

There are three possible combinations leading to the formation of sandhi. These are:

1. A vowel followed by a vowel is called *svara sandhi* or *samhita.*
2. A consonant followed by a consonant is called *vyanjan sandhi.*
3. A vowel and visarga followed by a vowel or consonant is called *visarga sandhi.*

Panini uses the word samhita only for vowel-vowel combinations. Other rules are just called consonant-consonant changes. Visarga sandhi under Panini's ashtadhyayi is actually nothing but changes associated with a final s or r in a word and is grouped by Panini with other consonant changes. Visarga rules are generally expressed as changes that occur when a particular vowel or group of vowels is followed by visarga which in turn is followed by various letters. Only in case when a hard consonant or a sibilant (i.e. Kar which denotes that letters K P C T W c t w v h y v r k p l S R s) follows is visarga actually employed even temporarily for the sequence of

changes leading to the final form. All other visarga rules are only the changes in final r.

2.3.1 Paninian Derivational System

Some of the sutras in Panini's grammar serve to define the technical terms used within the grammar of Panini. Some other sutras are nothing but rules of interpretation for other sutras. The sandhi rules in ashtadhyayi may be described as a series of ordered rewrite rules. The order in which rewrite rules are to be applied is as follows:

Precedence to a later rule The sutra viprawiReXe paraM kAryam dictates that in a situation where two sutras are applicable the later one is to be applied. As an example the sutra AxguNaH (Pa. 6.1.87) necessitates that if a letter a is followed by a letter o, then both of these two letters are replaced by the letter o. But the next sutra vrixXireci (Pa. 6.1.88) dictates that the two letters should instead by replaced by O. The sutra vrixXireci because of its later position in ashtadhyayi would be the one applicable.

Changes due to pUrvawrAsixXam The sutra pUrvawrAsixXam (Pa. 8.2.1) divides ashtadhyayi into two parts. The first part being called sapAxasapwAXyAyI[2] and the latter being called wripAxI [3]. This rule states that in a sequence of derivations from an initial form, if once a sutra in wripAxi is employed, any sutra preceding it should deem the change caused by the sutra in wripAxi as not having taken place.

Figure 1 shows a sequence of derivations. The sutra Pa. 8.3.19 causes the elision of y to take place. This change caused letter a to be followed by i. These two letters would have been, in normal course, replaced by a single letter e by Pa 6.1.87. But since the sutra Pa. 6.1.87 comes before Pa. 8.3.19

[2]literally meaning *seven chapters along with a quarter*
[3]literally meaning *three quarters*

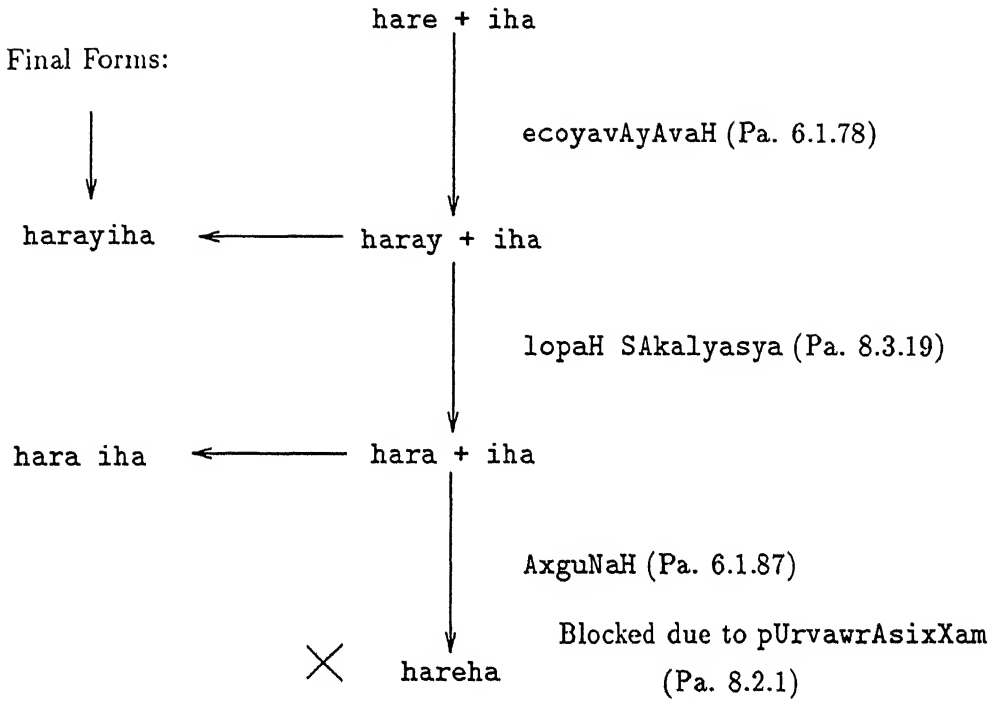


Figure 1: Illustration of pUrvawrAsixXam

(a sutra in wripAxI the change caused by the sutra Pa. 8.3.19 is not 'visible' to Pa. 6.1.87. Hence the final form remains hara iha.

Two more important characteristics are reflected in Panini's sutras for sandhi:

Reliance on abbreviations prawyAhAra: The whole of ashtadhyayi, in general and the sandhi rules in particular rely on the use of abbreviations prawyAhAra. These abbreviations are formed using the mAheSvara sUwra. Of the many possible abbreviations possible 42 are in use in ashtadhyayi. For example the abbreviation jaS denotes all the consonants j b g d x. Sandhi rules typically are of the form - "A Jal (i.e. any consonant except h y v r l or a nasal consonant) followed by JaS (i.e. J B G D X j b g d x) should be changed to a 'corresponding' jaS (i.e. j b g d x)." The 'correspondence' in the previous rule depends on the effort prayawna and the point of articulation Asya of the consonant to the replaced and the possible substitutes. This is further discussed below.

Reliance on effort and points of articulation Sandhi rules being nothing but rewrite rules must specify which letter is to be replaced by which other letter. This correspondence is established by Panini by saying sWAneZnwarawamaH which means that a substitute for a letter amongst a given set of choices should be one which is 'closest' in effort and point of articulation to the original letter. The 'closeness' of effort and articulation is further defined in siddhantakaumudi.

2.4 Summarization of all Sandhis

Tables 1 and 2 summarize the svara sandhis in Sanskrit. These two tables are to be interpreted like this:

- The letter on the left side denotes the letter that has to be present on the left side during sandhi.
- The letter on top denotes the letter that has to be present on the right side during sandhi.
- The corresponding entry of two letters in the table denotes the letter that will replace the letter on the left.
- Letters in () denote that both the letters (on the left and right) are to be replaced by this letter.
- Letters separated by comma denote a choice.

Table 3 summarizes vyanjan sandhis. The letters on top if present on left before a letter on the extreme right in the table is changed as indicated. Superscripts in this table denote the undergiven points:

1. The changes for uH are similar to iH
2. The changes for UH, EH,OH,EH,OH are similar to IH
3. The visarga may optionally assimilate before S,R or s. e.g. H+S $\rightarrow$ SS.
4. After n, S may remain unchanged.
5. n may also change to lz (nasalized vowel) when followed by l
6. n if preceded by a short vowel, is doubled before a following vowel.
7. *V* in the table denotes any vowel other than a.
8. Both the left and right letters change in this case. The change in the right letter is on the extreme right.

Tables 1, 2 and 3 summarizes all sandhis in Sanskrit at the *sentence level* i.e. between two words in a sentence. These tables have been adapted from [16] and [9]. It should be noted that there occur a lot of irregularities (not shown in the table) when forming compounds or verb inflections.

	a	A	i	I	u	U	q	Q	L	e
a	(A)	(A)	(e)	(e)	(o)	(o)	(ar)	(ar)	(al)	(E)
A	(A)	(A)	(e)	(e)	(o)	(o)	(ar)	(ar)	(al)	(E)
i/I	y	y	(I)	(I)	y	y	y	y	y	y
u/U	v	v	v	v	(U)	(U)	v	v	v	v
q/Q	r	r	r	r	r	r	(Q)	(Q)	(Q)	r
L	l	l	l	l	l	l	(Q)	(Q)	(Q)	l
e	(e)	ay, a	ay, a	ay, a	ay, a	ay, a	ay, a	ay, a	ay, a	ay, a
E	Ay	Ay, A	Ay, A	Ay, A	Ay, A	Ay, A	Ay, A	Ay, A	Ay, A	Ay, A
o	(o)	av, a	av, a	av, a	av, a	av, a	av, a	av, a	av, a	av, a
O	Av	Av, A	Av, A	Av, A	Av, A	Av, A	Av, A	Av, A	Av, A	Av, A
M	m	m	m	m	m	m	m	m	m	m
aH	(o)	ay, a	ay, a	ay, a	ay, a	ay, a	ay, a	ay, a	ay, a	ay, a
AH	Ay	Ay, A	Ay, A	Ay, A	Ay, A	Ay, A	Ay, A	Ay, A	Ay, A	Ay, A
iH	r	r	r	r	r	r	r	r	r	r

Table 1: Svara Sandhi

	E	o	O
a	(E)	(O)	(O)
A	(E)	(O)	(O)
i, I	y	y	y
u, U	v	v	v
q, Q	r	r	r
L	l	l	l
e	ay, a	ay, a	ay, a
E	Ay, A	Ay, A	Ay, A
o	av, a	av, a	av, a
O	Av, A	Av, A	Av, A
M	m	m	m
aH	ay, a	ay, a	ay, a
AH	Ay, A	Ay, A	Ay, A
iH	r	r	r

Table 2: Svara Sandhi (contd.)

k	t	w	p	f	M	n	aH	AH	iH^1	IH^2	
k	t	w	p	f	M	n	aH^3	AH^3	iH^3	IH^3	k, K, p, P, R, s
k	t	c	p	f	M	F^8	aH^3	AH^3	iH^3	IH^3	$S[S^8 \rightarrow C]^4$
k	t	c	p	f	M	MS	aS	AS	iS	IS	c, C
k	t	t	p	f	M	MR	aR	AR	iR	IR	t, T
k	t	w	p	f	M	Ms	as	As	is	Is	w, W
g	d	x	b	f	M	n	o	A	A	I	r
g	d	x	b	f	M	n	o	A	ir	Ir	g, G, d, D, b, B, y, v
g	d	j	b	f	M	F	o	A	ir	Ir	j, J
g	d	d	b	f	M	N	o	A	ir	Ir	d, D
g	d	l	b	f	M	M^5	o	A	ir	Ir	l
g^8	d^8	x^8	b^8	f	M	n	o	A	ir	Ir	$h[h^8 \rightarrow G, D, X, B]$
g	f	n	m	f	M	n	o	A	ir	Ir	n, m
g	d	x	b	f	m^6	n^6	o	A	ir	Ir	$a[a \rightarrow Z]$
g	d	x	b	f	m^6	n^6	a	A	ir	Ir	V^7

Table 3: Vyanjan Sandhi

The tables given will be able to handle most of the sandhis but they do not consider certain cases (which are covered by Panini's Ashtadhyayi. A few such cases are enumerated below:

1. As already stated above Panini explicitly does not mention visarga sandhi. Visarga sandhi originates from the changes that final r undergoes. What changes occur actually depend on how this final r originated. As an example the indeclinable r in punar has the word form punaH.

   punaH + ramawe → punA ramawe

   But

   rAmaH + ramawe → ramo ramawe

   This happens as the the visarga in rAmaH occurs from an intermediary form rAma ru as per Paninian derivational system. The u is elided the then the r gets converted into a visarga. The sandhi changes are different in this case then the visarga originated from a ru than when the visarga originated from a final r or an indeclinable.

2. In Ashtadhyayi the rules Pa. 8.4.46 to 8.4.52 specify the cases where reduplication of consonants occurring in conjuncts occurs. This reduplication is optional. An example illustrating this is shown in Figure 2.

   Actually there are four valid sandhi combinations when suXI and upAsya combine. These are suXyupAsya, suxXyyupAsya, suxXyupAsya, suXyyupAsya. These forms arise as y can also be optionally reduplicated. The Table 1 gives only one of these final forms viz. suXyupAsya

3. Pa. 6.1.95 omAfoSca gives rise to another 'irregularity'. An example of the change caused by this is shown below:

   SivAya + om → SivAyom

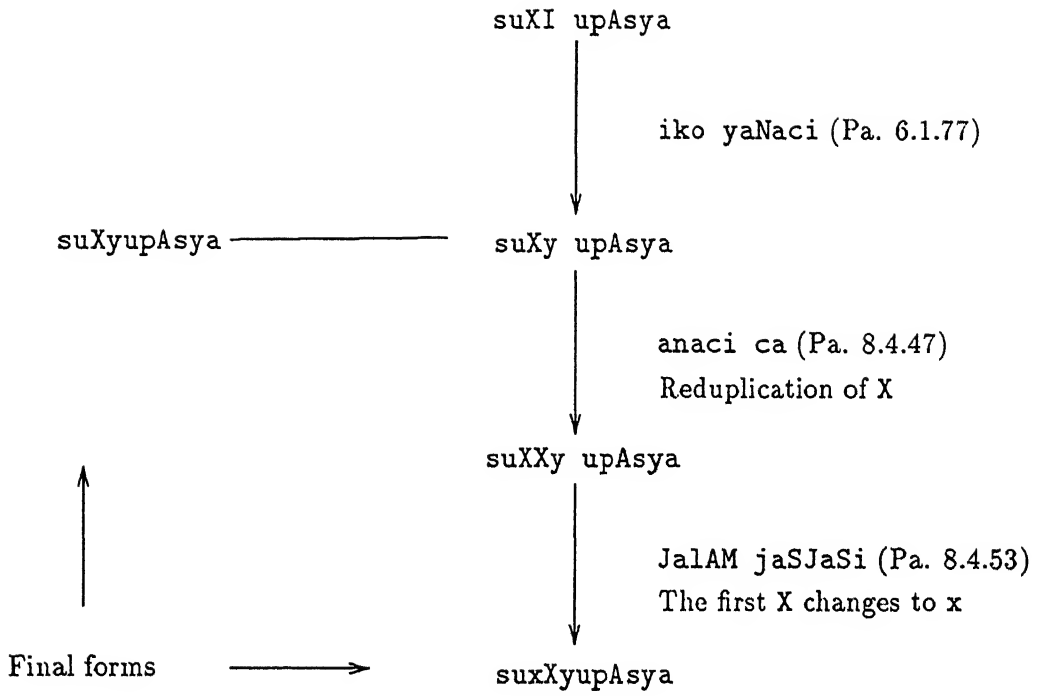


Figure 2: Illustration of Reduplication of consonants

and not SivAyOM as Table 2 would indicate. This is because of the modification of the rule when a is succeeded by the word om. Another 'irregularity' is shown below:

Siva + ehi → Sivehi

Thus Sivehi is the correct form instead of SivEhi. This is because the word ehi actually is a verb form with a prefix A. Thus in this case a rule is changed depending on the prefix which is occurring in the following word.

4. Panini defines a few words to be pragqhyaH. Vowel sandhi is not done when a pragqhyaH is succeeded by a vowel. A few examples of pragqhyaH are are follows:

   - A dual case affix ending in I or U or e is a pragqhya. Thus harI the dual form of the word hari is a pragqhya.

     harI + ewO → harI ewO (No change)

   - The inflected forms of the pronoun axas ending in I or U are pragqhyaH. A example is shown below for an inflected form amI of axas.

     amI + ISAH → amI ISAH (No change)

   - The final o of vocative singular praWamA samboXana is optionally pragqhya. In the example shown below viRNO is the vocative singular form of viRNu.

     viRNO + AgacCa → viRNO AgacCa (No change)

5. There are special (optional) rules for the word nQn followed by the letter p. Thus nQn followed by pAhi can take the form dn nQnpAhi (the regular form) but this is only one of the allowed five forms.

As can be seen from above examples while mostly the sandhi rules are indeed simple rewrite rules (as shown in Tables 1,2 and 3) a precise description is only possible after taking into account the morphology of words on which sandhi is being done.

Chapter 3

PC-KIMMO

PC-KIMMO is a computer program that uses a linguist's description of the phonology and morphology of a natural language to recognize and generate words in that language. It is a new implementation for microcomputers of a program dubbed KIMMO after its inventor Kimmo Koskenniemi, a Finnish computational linguist. Implementations of PC-KIMMO are available on Unix System V, BSD, SCO, IBM PC and on Macintosh.

The theoretical model of phonology embodied in KIMMO [1]is called two-level phonology. In the two level approach, phonology is treated as the correspondence between the *lexical level* of underlying representation of words and their realization on the *surface level.* The two functional components of KIMMO are the *generator* and the *recognizer.* The generator accepts as input a lexical form, applies the phonological rules, and returns the corresponding surface form. It does not use the lexicon. The recognizer accepts as input a surface form, applies the phonological rules, consults the lexicon, and returns the corresponding lexical form with its gloss. Figure 3 shows the main components of the KIMMO system.

A KIMMO description of a language comprises of two components:

[1]From now onwards unless specifically mentioned the word *KIMMO* is used for *PC-KIMMO.*

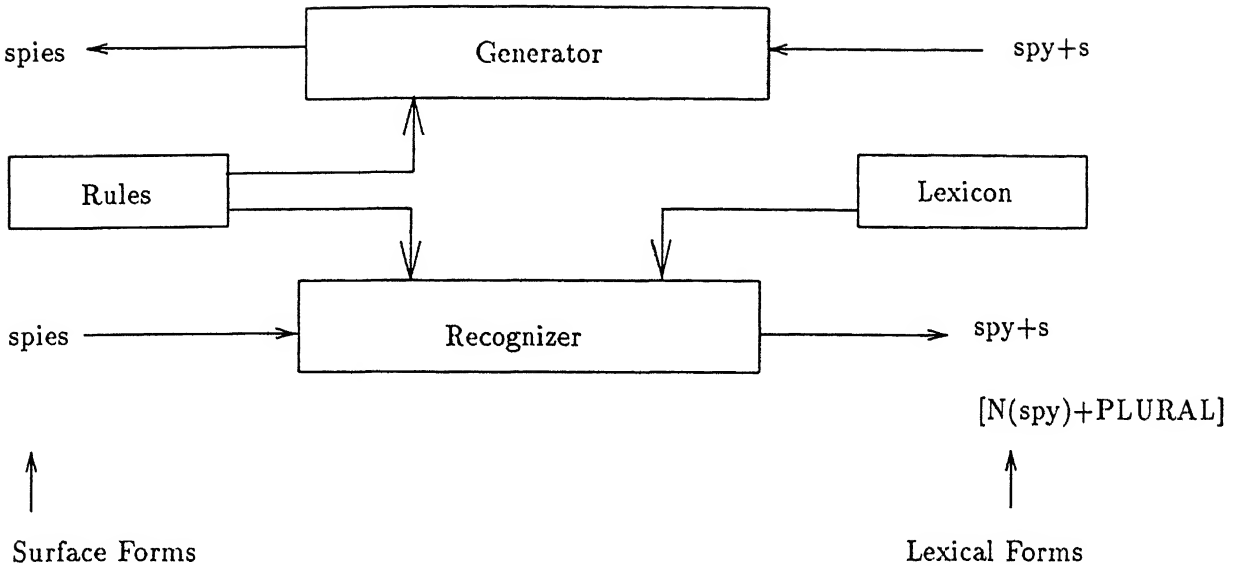


Figure 3: General Structure of KIMMO

1. The rule component comprising the alphabet and the phonological rules.
2. The lexical component which lists lexical items (words and morphemes) and their glosses and encodes the morphotactic constraints.

3.1 The lexical component

In KIMMO, the lexical component is encoded using a *finite state machine.* A finite state machine (or *FSM* for short) is defined as follows:

Definition 3.1 *finite state machine is a 5-tuple* $M = (Q, \Sigma, \delta, q_0, F)$ *where* Q *is the (finite)* set of states *of* M, Σ *is the finite* alphabet, δ *is the* transition function *which maps* $\Sigma \times Q$ *to* Q, q_0 *is the* initial state *of the finite state machine and* F *is the* set of final states.

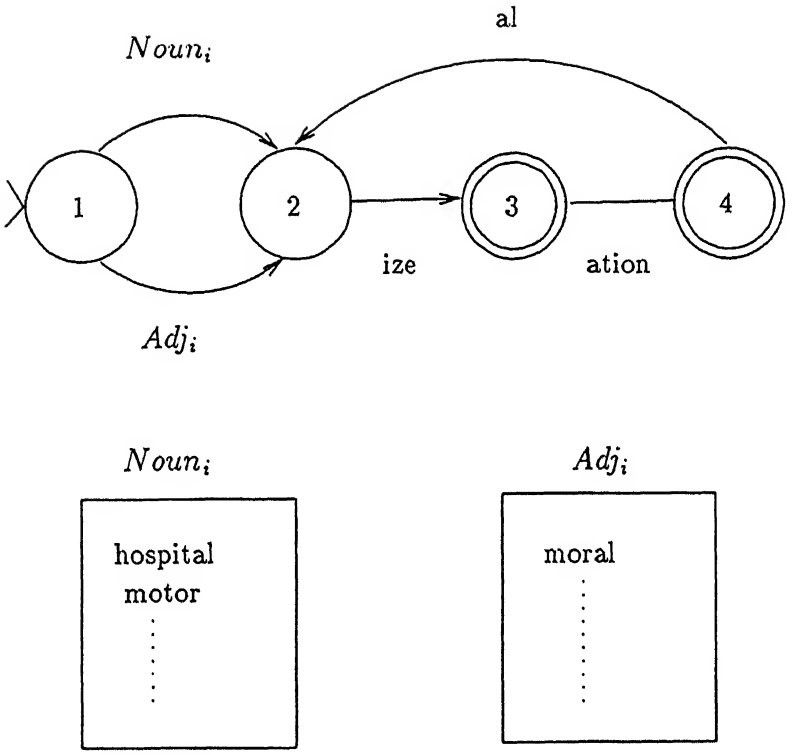


Figure 4: An example FSM encoding English morhpotactics

An example FSM encoding a very small fragment of English morphotactics is shown in Figure 4. Also in the figure are shown the sublexicons $Noun_i$ and Adj_i. It can be seen that the FSM permits the putting together of morphemes *motor*, *ize* & *ation* as *motor+ize+ation*. Thus the lexical component in KIMMO specifies how the various morphemes in a particular language being modeled can be put together.

The lexicon component thus comprises a finite state machine whose arcs represent sublexicons for the language. Any path from the initial state to the final state in the FSM for the lexicon thus defines a valid combination how the various morphemes can be put together. The challenge in representing the morphotactic constraints in a particular language for KIMMO lies in divising appropriate finite state machine and appropriate sublexicons (which label the arcs in the FSM) such that an accepting path for the FSM represents a valid putting together of the morphemes for the language. Further the description of the finite state machine used for encoding the morphotactic constraints needs to be written in the input file format acceptable by KIMMO. This file format is described by Antworth [1]. In an actual implementation of KIMMO a trie can be used to encode the morphotactics of a language.

KIMMO also permits to add a *gloss* entry for each item in a sublexicons. While this may be same as the lexical entry itself, it is possible to put some more information as a gloss. KIMMO directly does not use any of the information in the gloss except that the gloss may present more information to a user. In the example presented above the glosses of *motor*, *ize* and *ation* may be *Noun(motor)*, *Suffix(ize)*and *ation* respectively. When the user working with KIMMO asks for the recognition of the word *motorization* the result of the morphological analysis is presented as *motor+iza+ation*. Besides the glosses of these morphemes are also concatenated and presented as *Noun(motor)+Suffix(ize)+Suffix(ation)*. This information in the gloss may also be used by another program running KIMMO as a subprocess (or by a program using library calls of KIMMO using the programming interface of KIMMO).

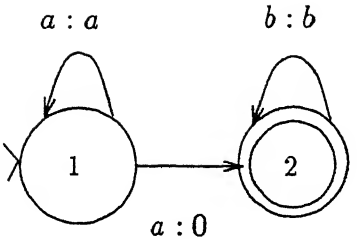


Figure 5: FST accepting the language $(a : a)^*\ a : 0\ (b : b)^*$

Slightly different terminology has been used traditionally in literature. *Alternations* in literature refer to the states of the FSM of the lexical component. *Continuation classes* refer to the sublexicons (which are denoted by arcs in a diagram for FSM).

3.2 The Rule Component

Phonological rules are implemented in KIMMO as *finite state transducers.*

Definition 3.2 *Finite state transducer is a 5-tuple* $M = (Q, \Sigma, \delta, q_0, F)$ *where* Q *is the (finite)* set of states *of* M, Σ *is the finite* alphabet *each of which comprises of a pair of two symbols,* δ *is the* transition function *which maps* $\Sigma \times Q$ *to* Q, q_0 *is the* initial state *of the finite state machine and* F *is the* set of final states.

The finite state transducer, *FST* for short, unlike finite state machine accepts a language over a pair of symbols in the alphabet. Another way in which FST's can be interpreted is as a relation from strings to strings. While an FSM defines a *regular language*, the FST defines a *regular relation.* In a manner similar to FSM's [11], the FST's can be written as a regular expression in terms of an alphabet of paired characters. The Figure 5 shows an example FST which accepts the language $(a : a)^*(a : 0)(b : b)^*$. Of the infinitely many strings that this FST will accept three example string tuples are shown which are accepted by this FST.

$$< aaabbb, aabbb >$$

$$< a, \epsilon >$$

$$< abb, bb >$$

In KIMMO, it is possible that the string tuple accepted by it does not have strings of equal size. This is the actually the case in the examples cited above. This is handled by positing a special character *NULL*. This character NULL is treated as exactly any regular symbol as far as the FST is concerned. Thus the FST actually accepts a pair of equal length strings. Thus for example the first string tuple is actually accepted the FST as the pair $< aaabbb, aa0bbb >$ which is a pair of equal length strings, each of whose length is 6. The FST will when accepting this pair of strings will operate as shown below.

State	Symbol Pair Accepted	New State
1	$a : a$	1
1	$a : a$	1
1	$a : 0$	2
2	$b : b$	2
2	$b : b$	2
2	$b : b$	2

Since 2 is a final state in the FST this string over the pair of symbols is accepted.

Thus while usually the literature (e.g. [13],[22]) mentions only two levels in two level morphology, actually four separate levels exist [18]. These are shown in Figure 6.

The *surface form* is the final form of the word while the *Lexical Form* is the analysis that KIMMO will yield when asked to analyze the word *moved*. The *lexical tape* is the concatenation of the lexical forms possibly with the addition of nulls while the *surface tape* contains the symbols of the surface form possibly with the addition of NULLs. As said earlier the NULLS are treated by FSTs as any other character.

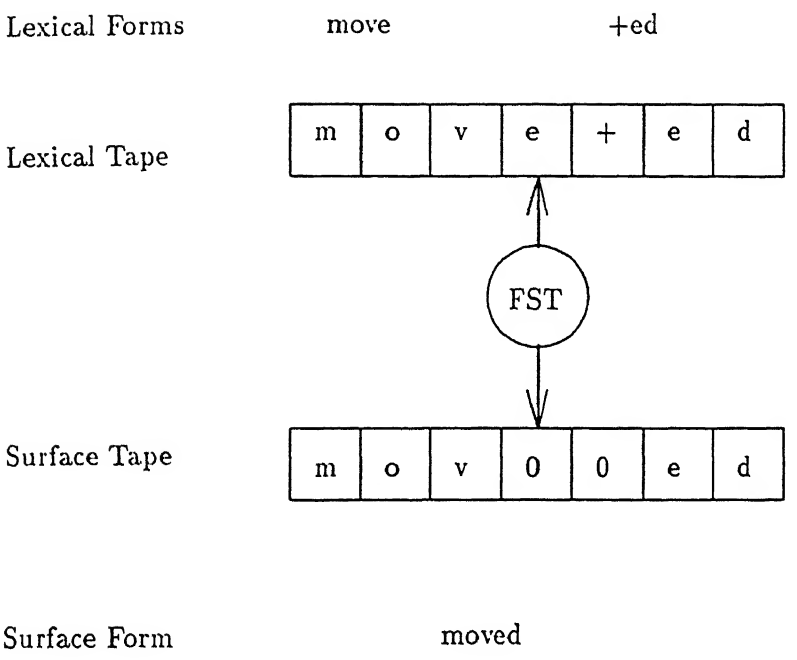


Figure 6: The four levels in KIMMO

Thus the lexical tape and surface tape will contain equal number of symbols. The FST thus defines a possible pairs of equal length sequences (the two tapes). Using the convention of regarding NULLs as ordinary characters it is possible to relate the lexical and the surface forms which may not be equal in length.

While the FSTs can always be represented by the diagrammatic notation as in Figure 4, as a matter of notational convenience they are usually represented in the form of a table. Thus the FST in Figure 4, can be represented as

	a	a	b
	a	0	b
1.	1	2	0
2:	0	0	2

The top two rows represent the possible alphabet pairs $a : a$, $a : 0$, $b : b$. The first column denotes the states. A dot following a state denotes that the state is a non-final state while a colon following a states denotes that it is a final state. The

state 0 denotes a sink state. Many a times a special symbol @ is employed to denote ANY character. As an example, suppose that the symbols $x:y$, $y:y$, $y:@$, $@:@$ are used in one FST. The symbol $y:@$ would denote all the possible symbol pairs in the alphabet of FSTs where the lexical form is y minus the explicitly mentioned pair $y:y$. The symbol $@:@$ would mean all possible symbol pairs in the alphabet of FSTs but leaving out the specifically mentioned symbols $x:y$, $y:y$ and $y:@$.

Another abbreviatory device used to denote FSTs is to use sets of alphabets. Thus the symbol $A:a$ where the set $A = \{a, b, c\}$ would mean all the symbol pairs $\{a:a, b:a, c:a\}$. The symbol $A:B$ where A is as previously defined and B is $\{c, d, e\}$ will denote all symbols $\{a:c, b:c, c:c, a:d, b:d, c:d, a:e, b:e, c:e\}$.

To use KIMMO for any language we need to design appropriate lexical forms (i.e. the forms in which we would like the analysis to be given), the alphabet of the FSTs and the FSTs themselves. All three put together would thus define a relation from the lexical forms and the final surface forms.

In practice instead of a single large FST encoding all the phonological rules of a language, a number of FSTs are used. These FSTs are operated in parallel on the lexical and surface tapes i.e. the same tape(s) will be fed to all the FSTs and all the FSTs are constrained to move their heads in unison across the length of the tape(s). Each of the FST represents a regular relations of equal length strings. The language accepted by all the FSTs together thus will be the intersection of all these relations.

3.3 How KIMMO works ?

Once the lexical component and the rules component are ready, KIMMO can be used to both recognize (i.e. take in a surface form and given out the possible corresponding lexical form(s)) as well as generate (i.e. take in a surface form and given out the possible corresponding surface form(s)). While in case of generation the surface form is available, in case of recognition the lexical form is given. Another

difference between these two processes is that the lexical component is not used in case of generation.

3.3.1 Generation

Assume that we have n FSTs for a given phonological description. The state of all the FSTs can be denoted by a *state vector* consisting of a vector of size n. The state of the ith component of the state vector will denote the ith FST. All the components of this vector are initially 1 (the start state of each of the FSTs). The generator component attempts to move from the beginning of the lexical form to its end. In each step it attempts to posit a symbol pair (i.e. two characters, one for the lexical tape, the other for the surface tape) corresponding to next available character in the surface form. Suppose a character α is the next available character on the lexical form. If the symbol pair alphabet of the FST is Σ, he next symbol pair on the lexical and the surface tapes can be any element in the set S defined below.

$$S = \{< \alpha, \beta > \mid < \alpha, \beta > \in \Sigma\} \cup \{< 0, \beta > \mid < 0, \beta > \in \Sigma\}$$

The first component in the right hand side of the equation above comes if the next available character from the lexical form is posited as the next character in lexical tape. The latter component arises if we posit a NULL character at this point on the lexical tape. KIMMO then provides this symbol pair to all the FSTs and each of the FSTs are made to undergo a transition. This will lead to a change in state vector. If any of the component of state vector is now zero (i.e. that particular FST goes into the sink state) KIMMO backtracks. Otherwise it continues with the next available character on the lexical form. Thus by repeated backtracking, till the end of all characters from the lexical form, it produces all possible surface tape (and thus surface forms by deleting NULLs from the surface tape). If all the FSTs are in final states at this juncture the surface form generated is treated valid.

Essentially in this procedure KIMMO tries to find a path, in each FST, from the initial state of the FST to its final state by backtracking. The possible surface tapes are found out by using the lexical form and from the surface tape the surface form can be found out.

3.3.2 Recognition

The recognition procedure is very similar to the generation part. The difference lies in the fact that here the surface form is known rather than the lexical form. Using backtracking, as above, the corresponding lexical form is found out. The lexical component of KIMMO is also used in recognition. Whenever a character of the lexical form is posited it is checked in the lexical component to ensure that it also adheres to the morphotactic constraints stipulated by the lexical component.

3.4 Two level rules

While it is possible to directly write FSTs for the rules component, a different notation is usually followed as an intermediate step before obtaining the FSTs. This is the *two level rule* notation.

The two level rule notation permits rules of the following four kinds:

$$a : b \quad \Rightarrow \quad LC\_\_\_\_RC \tag{1}$$

$$a : b \quad \Leftarrow \quad LC\_\_\_\_RC \tag{2}$$

$$a : b \quad / \Leftarrow \quad LC\_\_\_\_RC \tag{3}$$

$$a : b \quad \Leftrightarrow \quad LC\_\_\_\_RC \tag{4}$$

A rule of the form (1) is called *context restriction rule.* A rule of the form (2) is called *surface coercion rule.* Rules of the form (3) and (4) are respectively called *exclusion rule* and *composite rule.* *LC* denotes the Left correspondence part and *RC* the right correspondence part. Both LC and RC are regular relations. Either of these two contexts can be empty which would mean that it could be anything. The semantics of these rules are described below.

Context Restriction rule The context restriction rule (1) states that the letter a on the lexical tape can correspond to a letter b on the surface tape but wherever this correspondence is present the left side of the tapes must match the given context *LC* and the right side of the tapes must match the given context *RC*. In effect this rule limits the places where the correspondence $a : b$ can be found.

Surface Coercion rule The surface coercion rule (2) states that wherever on the tapes the letter a on lexical tape is preceded by the context *LC* and followed by RC the surface tape must contain the letter b. Thus this rule mandates (coerces) the surface character under match left and right contexts. Note that this rule does not rule out the occurrence of some other character say c between *LC* and *RC*. It only states mandates the occurrence of b on the surface side when the lexical tape contains a and the context matches.

Exclusion Rule While the surface coercion rule states what the letter a should correspond to in the given context, the exclusion rule (3) that surface side must not be b in the given context. This rule does *not* rule out the possibility of c in this context. Thus the symbol pair $a : c$ will be valid as far as this rule is concerned.

Composite Rule This is a combination of context restriction rule and the surface coercion rule. The composite rule (4) states that the symbol pair $a : b$ must occur in the given context and wherever this symbol pair occurs the given context must also be present.

The use of abbreviatory devices for notation of FSTs has been described previously in Section 3.2. These abbreviatory devices are used in two level rule notations too. Thus, we could have rules like

$$c:a \Rightarrow x:y\_\_\_\_B:C$$
$$c:c \Rightarrow x:@\_\_\_\_$$

B and C are sets and @ is the ANY character in the above rules.

3.4.1 Designing the alphabet

For KIMMO to work the possible alphabet (of symbol pairs which will be accepted by FSTs) needs to be provided. This alphabet will comprise two parts, the ordinary correspondences where every alphabet in the language is mapped to itself to form a symbol pair. This is known beforehand as the alphabet of the language whose phonology is to be modeled is known. But the special correspondences are known only after the two level rules have been written down. Thus for example in writing rules for sandhi analysis we write the rule given below.

$$h:D \Leftrightarrow t:t \; +:0\_\_\_\_$$

This rule arises from the sandhi rule "h should be replaced by D if preceded by t". Thus the special correspondence $h : D$ would have to be added to the symbol pair alphabet of the FSTs.

3.4.2 KGEN

While it is possible to hand compile the two level rules [1] if the rules are simple enough, hand compilation becomes increasingly difficult with more complex ones. Especially if numerous rules need to be written down it becomes very tedious to hand compile these rules to FSTs. KGEN [15] is one such rule compiler. It is available from Summer Institute of Linguistics, Dallas, Texas and is in public domain. The author of this thesis has employed KGEN for the compilation of two level rules to FSTs for this project.

3.5 Complexity results

The generation and recognition problems for KIMMO have been shown to be NP-hard by Barton [4, 5]. This has been established by reducing these problems to Boolean satisfiability problems SAT and 3SAT which involve decideding where a given Boolean formula in Conjunctive Normal Form (CNF) has a satisfying truth-assignment. If NULL characters are disallowed these problems have been shown to be NP-complete (i.e. it is known that these restricted problems are in NP). If unrestricted NULL characters are allowed these problems may be harder than NP-complete problems. Barton & Berwick show that they are also PSPACE-hard - i.e. at least as hard as any problem that can be solved in polynomial space.

These results at first would suggest that KIMMO may be a very costly approach. Koskenniemi and Church [14] have argued that in practice KIMMO turns out to be very efficient. They present data demonstrating KIMMO's use on a Finnish corpus of 400,000 tokens. Their results are that the number of analysis steps is given approximately by

$$Steps = 2.43 \times WordLength + Constant.$$

This stipulates that this approach is perhaps not as costly as the complexity results augur it to be.

Chapter 4

Sandhi rules to Two-level Rules

By now the structure of the sandhi rules has been discussed (Chapter 2). The analysis of phonology and morphology of many languages has already been attempted using KIMMO. Notably for English [2] large (more than 20,000 lexicon entries) lexicons and rules have been built. Besides the program PC-KIMMO is distributed with example (toy) solutions of phonological analysis of Hebrew, Tagalog, Finnish, Japanese, Mende, Turkish and Zoque. KIMMO has indeed been by far the most successful model of computational morphology [22]. Lot of work has indeed been done on the morphology/phonology of various languages using two-level morphology as the underlying formalism.

4.1 Taking Stock

Two of the possible solutions for analysis of sandhi can be enumerated as below:

Modeling Paninian Sutras directly One method can be to model the Paninian aphorisms directly a la Fonol [7]. But this presumes developing a precise

computational model for these sutras. The advantage of this method would that it, being a mirror of Paninian rules, would be able to capture the rules of Sanskrit very accurately. But this does assume the statement of rules in ashtadhyayi in an unambiguous mathematical form.

Using Existing Formalisms Existing formalisms include Fonol [7] and KIMMO itself. Fonol tries to model the rules of traditional generative phonology i.e. a sequence of ordered rewrite rules of the form $A \rightarrow B/\alpha\_\_\_\_\beta$ which means that A is replaced by B in the given context. Unlike KIMMO any number of intermediate levels are permitted. Using Fonol assumes the statement of Sanskrit rules in the form of rules used in generative phonology. Another formalism that exists is KIMMO itself. Using two-level rules necessitates the rewriting of sandhi rules in this formalism.

4.2 The Method Used

Given the choice of above methods the method used within the scope of this thesis is writing sandhi rules in terms of two-level rules. For testing it PC-KIMMO was used. The reasons behind this choice are discussed below.

As discussed in Section 3.5 we do not have a theoretical guarantee that the recognition problem in KIMMO will be efficient in the sense that we have a polynomial time algorithm [4, 5]. But as has been reported by Church and Koskenniemi [14] KIMMO does work efficiently in practice. In fact Church and Koskenniemi have shown a linear relation of the time used for morphological analysis with increasing word-length for a large corpus of Finnish words. Thus, efficiency is a very strong point in favour of KIMMO.

In comparison, Fonol models ordered rules of Generative Phonolgy directly. This can be expected to more time consuming. Similarly, the modeling of Paninian rules

(along the lines they are organized in Panini) can also be expected to be time consuming. Also this assumes precise specification of Paninian sutras in mathematical form.

Further, KIMMO is available in the public domain and lot of work has already been done to attempt to model the morphology/phonology of natural languages using KIMMO. As a result, the use of KIMMO is well documented [1] and also example large size morphological analyzers for various languages are available (e.g. [2]).

Sandhi rules in tabular form (Tables 1, 2 and 3 in Chapter 2) have been converted to two-level rules. The process of conversion to two-level rules is described in the next section.

4.3 Rules Component

The essential mechanism in being able to write rules using two-level morphology/phonology is to propose two-level correspondences for each rule such that the proposed two-level accurately models the rule in the natural language. But in this process the fact that only two-levels are available is to be taken care of. While rules in Paninian framework (or traditional phonology) admit any number of intermediate forms, the mapping from the lexical tape to the surface tape (and vice-versa) has to be direct through an intervening finite state transducer. By using the rules in tabular form as in Tables 1,2 and 3 (in Chapter 2), this is ensured as two-levels are directly available to us – the lexical side being the rules on the top and left sides and the surface side being the entry inside the table. Also once these rules have individually been written it has to be ensured that they do not conflict with each other i.e. a two-level rule resulting from one particular mapping of lexical tape to surface tape must not rule out another valid mapping of lexical tape to surface tape.

A representative sample of conversion of sandhi rules to two-level rules is considered below. These illustrate how sandhi rules can be converted to two-level rules.

4.3.1 Handling deletion

Consider the sandhi rule "A visarga followed by a is dropped if followed by any vowel except a." The following two-level correspondence can be used for this rule:

Lexical Tape	a	H	+	V'
Surface Tape	a	0	Blank	V'

In the above two level correspondence the 'a' on the lexical tape corresponds to 'a' on the tape, the visarga ('H') to the NULL symbol, the + to the Blank symbol and any vowel except 'a' (denoted by V') is mapped to itself (i.e. 'A' on the lexical tape in place of V' should correspond to 'A' on the surface tape). This rule for the deletion of visarga can be represented as a two-level rule shown below:

$$H : 0 \Leftrightarrow a : a\_\_\_\_\_+ : Blank\ [A|i|I|u|U|e|E|o|O|q|Q] \tag{5}$$

[1]

This rule would imply that the combination $H : 0$ is allowed only in the context when $a : a$ is on its left and followed by a $+ : Blank$ and then by any vowel-pair except $a : a$. This rule is the combination of the following two rules:

$$H : 0 \Leftarrow a : a\_\_\_\_\_+ : Blank\ [A|i|I|u|U|e|E|o|O|q|Q|L] \tag{6}$$

$$H : 0 \Rightarrow a : a\_\_\_\_\_+ : Blank\ [A|i|I|u|U|e|E|o|O|q|Q|L] \tag{7}$$

The operation of the surface coercion rule (2) can be understood by looking at the finite state transducer equivalent to this rule. This FST is shown in Figure 7. The

[1]We here use an abbreviation permitted by two level rules. The symbol x (left unpaired) in a context means same as $x : x$. Thus the said rule is an abbreviation of $H : 0 \Leftrightarrow a : a\_\_\_\_\_+ : Blank\ [A : A|i : i|I : I|u : u|U : U|e : e|E : E|o : o|O : O|q : q|Q : Q]$

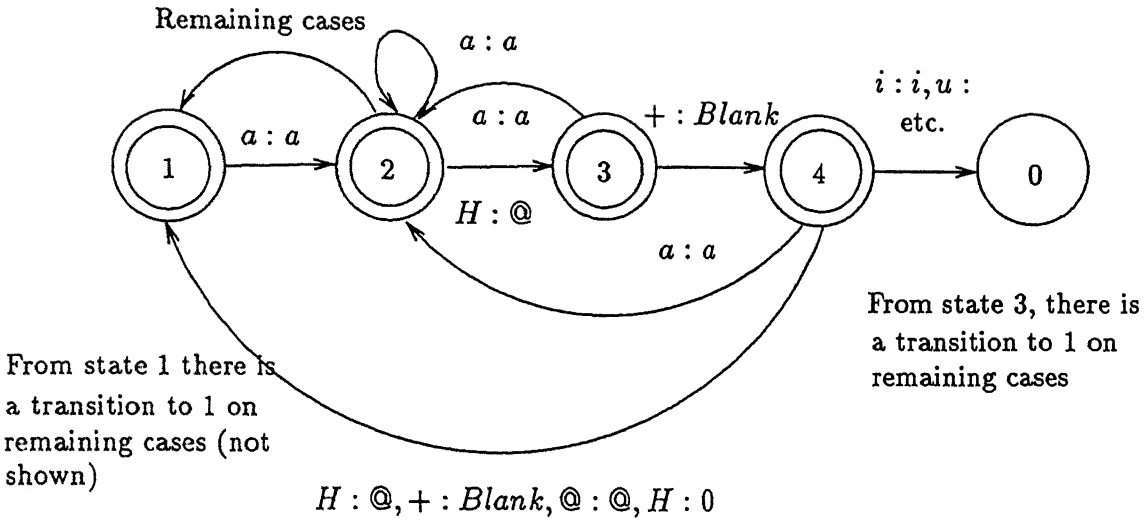


Figure 7: Operation of Surface Coercion Rule

rule operates like this — only on the receipt of symbol $a : a$ the FST enters the state 2. If in state 2 it finds a visarga mapped to any symbol except 0 (i.e. on receiving $H : @$) it enters state 3. Now only if receives the symbol $+ : Blank$ it enters the state 4. Otherwise it continues in state 1. If in state 4 it receives a vowel-vowel pair it enters the sink state. Otherwise it continues in state 1.Thus, the state 1 denotes the normal accepting state, the state 2 denotes that a symbol $a : a$ has been received and the state 3 denotes the condition that a $H : @$ has been received after receiving $a : a$. State 4 denotes the state after receiving the sequence of symbols $a : a$, $H : @$ and $+ : Blank$. Thus the FST in Figure 7 rejects the input if and only if H is mapped to anything else but 0 when it is preceded by $a : a$ and followed by $+ : Blank$ which is further followed by a vowel (except 'a') mapped to itself. In tabular notation the FST in Figure 7 would be written as:

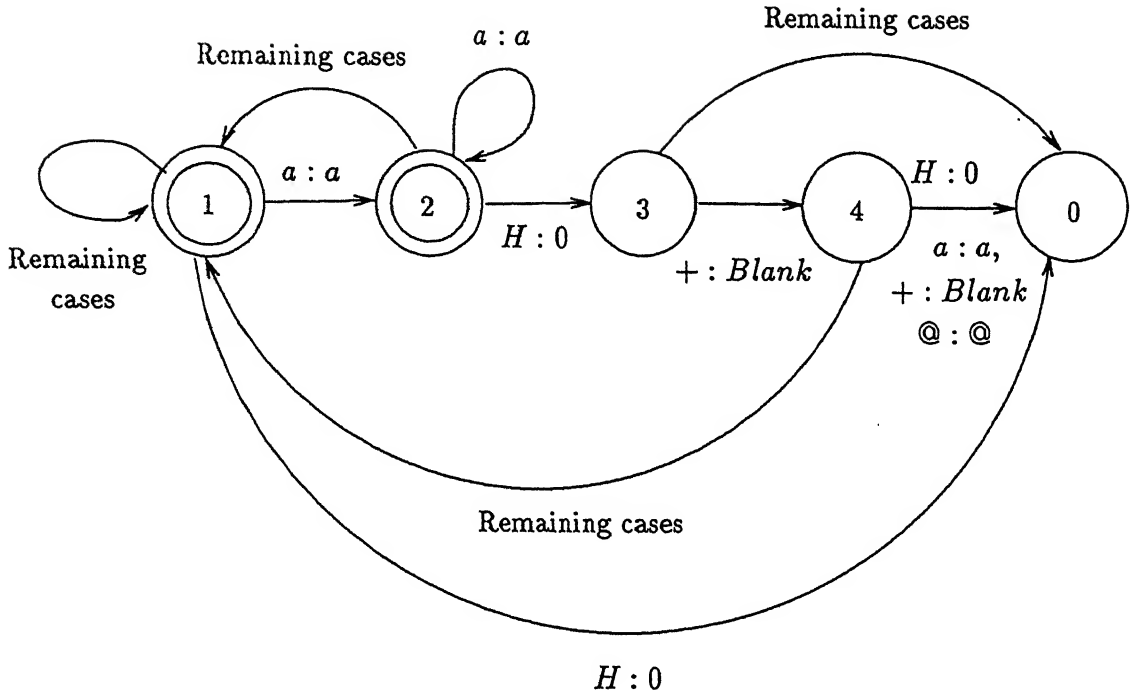


Figure 8: Operation of Context Restriction Rule

	a	H	+	@	H	A	i	I	u	U	e	E	o	O	q	Q	L
	a	@	Blank	@	0	A	i	I	u	U	e	E	o	O	q	Q	L
1:	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2:	2	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3:	2	1	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4:	2	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

The finite state transducer in Figure 8 is equivalent to the context restriction rule 3 . The rule operates like this — the FST remains in initial state 1 unless the symbol $a : a$ is received. But if the symbol $H : 0$ is received it enters the sink state 0 (because the left context gets violated if $H : 0$ is received without first receiving $a : a$). In the state 2, the FST enters state 3 only on receipt of symbol $H : 0$. Otherwise it returns to the symbol 1. But if $a : a$ is received it remains in state 2. In state 3 it enters state 4 on receipt of $+ : Blank$ otherwise it goes to sink state. In state 4, the FST enters sink state on receipt of anything else but a vowel-vowel

pair and on receipt of a vowel-vowel pair it goes back to state 1. The state 1, thus denotes the normal state. The state 2 denotes the condition when $a : a$ has been received. State 3 denotes the condition when $H : 0$ is received after symbol $a : a$. State 4 denotes that $a : a$, $H : 0$ and $+ : Blank$ have been received in this order. If the FST ends in any state but 4 it does not violate the context restriction rule. So, states 1, 2 and 3 are made final states. In tabular form the above FST can be represented as below:

	H	a	+	@	a	A	i	I	u	U	e	E	o	O	q	Q	L
	0	a	Blank	@	a	A	i	I	u	U	e	E	o	O	q	Q	L
1:	0	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2:	3	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3.	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4.	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

The composite rule 1 operates as a conjunction of both the conditions 2 and 3. The combination of the two FSTs in Figures 7 and 8 operating is parallel is equivalent to the composite rule 1.

4.3.2 Handling Insertion

Consider another sandhi rule — "If after a short vowel a C is present a c is inserted before this C."This rule involves the insertion of c. To handle the insertion we posit the following two level correspondence:

Lexical Tape	V'	+	0	C
Surface Tape	V'	0	c	C

This correspondence can be handled by the following two rules:

$$0 : c \Rightarrow [a|i|u|q|L] + : 0\_\_\_\_C : C \tag{8}$$

$$0 : c \Leftarrow [a|i|u|q|L] + : 0\_\_\_\_C : C \tag{9}$$

These rules operate in a manner analogous to the rules described above. The FST for the surface coercion rule goes into sink state if if receives the left context followed by 0 mapped to anything else but c which is further followed by $C : C$. This will ensure the restriction on the surface character corresponding to 0. On the other hand the surface coercion rule operates by not accepting the cases when either $0 : c$ comes before the left context or when the right context does not follow the left context and the symbol pair $0 : c$.

4.3.3 Handling Assimilation

Assimilation is a phonological process in which on sound segment influences another so that the sounds become more alike, or even identical. An example of a sandhi rule of this kind is "k followed by any vowel is changed to g". For this rule the following two-level correspondence is proposed.

Lexical Tape	k	+	V
Surface Tape	g	0	V

This correspondence can be handled by the following two rules:

$$k : g \Rightarrow \_\_\_\_+ : 0\ [a|A|i|I|u|U|e|E|o|O|q|Q|L] \tag{10}$$

$$k : g \Leftarrow \_\_\_\_+ : 0\ [a|A|i|I|u|U|e|E|o|O|q|Q|L] \tag{11}$$

As earlier, the surface coercion rule (7) ensures that k is indeed transformed into g in appropriate context while the context restriction rule (6) ensures that k can be mapped into g only under the appropriate context.

4.3.4 Handling a mix of phenomena

The Sections 4.3.1 to 4.3.3 talk about the phenomena of deletions, insertions and assimilations in isolation. In practice, in a single sandhi rule more than one of these phenomena may occur. In fact, even in the rules stated above this happens. For example in Section 4.3.3 while k is mapped to g no restriction is placed on the occurance of + : 0. Thus KIMMO when given a string to recognize would go into an infinite loop for it will posit an infinite sequence of the character '+' on the lexical tape while taking the corresponding character on surface tape as '0'. This can be prevented by positing an additional rules:

$$+ : 0 \Leftarrow k : g\_\_\_\_[a|A|i|I|u|U|\epsilon|E|o|O|q|Q|L] \quad (12)$$

$$+ : 0 \Rightarrow k : g\_\_\_\_[a|A|i|I|u|U|\epsilon|E|o|O|q|Q|L] \quad (13)$$

Thus for any sandhi rule the procedure to convert sandhi rule is:

1. Write the two-level correspondence desired.
2. For each of the special character correspondences [2] in the two-level correspondence write both the surface coercion rule and also the context restriction rule.

4.3.5 Maintaining consistency

The procedure in the previous section ensures the FSTs are able to model the sandhi rules. While the FSTs written for a particular sandhi rule reflect that rule,

[2] e.g. $k : g$ is a special correspondence while $k : k$ is an ordinary correspondence. Each of the special correspondence dictates that the surface form must be coerced appropriately and also the context where it occurs is also limited

it is possible that when all such FSTs for all the rules are taken together some formations are ruled out. For example: k changes to g before the consonants r, g,G,d,D,b,B,y,v,j,J,l,h,n,m.

This would give the rule:

$$k : g \Rightarrow \_\_\_\_\_+ : 0\ [r|g|d|D|b|b|y|v|j|J|l|h|n|m] \tag{14}$$

$$k : g \Leftarrow \_\_\_\_\_+ : 0\ [r|g|d|D|b|b|y|v|j|J|l|h|n|m] \tag{15}$$

The rules 6 and 10 are actually incompatible. The rule that actually should be put forth should be:

$$\begin{aligned} k : g \ \Rightarrow . \ & \_\_\_\_\_+ : 0\ [r|g|d|D|b|b|y|v|j|J|l|h|n|m] \\ & |\_\_\_\_\_+ : 0\ [a|A|i|I|u|U|e|E|o|O|q|Q|L] \end{aligned} \tag{16}$$

The symbol | denotes the disjunction of the two contexts. Thus after converting all sandhi rules into two-level rules, another step of finding and ironing out inconsistencies has to be done. This is done by grouping all the contexts of each of the special correspondences into a disjunction.

4.3.6 Multiple final forms

In the cases where multiple final forms are possible (this is possible due to optional rules or where rules actually provide two forms) the conversion process to two-level rules is similar. In such cases two or more two-level correspondences will occur. Each of the correspondences can then be converted to two-level rules by the process described above.

4.3.7 Finding the alphabet

The alphabet for the particular two-level set for the sandhi-rules depends on the two-level correspondences used. The ordinary correspondences in the alphabet of FSTs are just each of the character in alphabet of the language (in our case Sanskrit) mapped to itself. The special correspondences are known only after all the rules are known. Thus after all the rules are known, the special correspondences employed in each rule are added to the already known ordinary correspondences to give the full alphabet.

4.4 Lexical Component

All the details regarding the rules component have been specified above leaving out only the lexical component. The lexical component, as already described in Chapter 3 is organized as a finite state automaton. The arcs in this FSM are the sublexicons. KIMMO requires that the lexical component use a special lexicon called INITIAL to branch out from the initial state of this FSM. This FSM is shown in Figure 9. The FSM uses two sublexicons `SANSWORD` which is a list of Sanskrit words and COMBINER which consists of a single symbol 0. This FSM will permit any sequence of Sanskrit words separated by the character '+'. The lexical component is used by KIMMO only during the recognition phase. Each character thus posited during the recognition phase is checked for against the FSM of the lexical component. As soon as a character is generated that does not match with the FSM, it is known that further characters need not be generated. In particular, in sandhi analysis if a word does not exist in lexicon it cannot participate in sandhi (or in other words the sandhi analysis is incorrect unless each of the words that participate in sandhi is a valid word in lexicon).

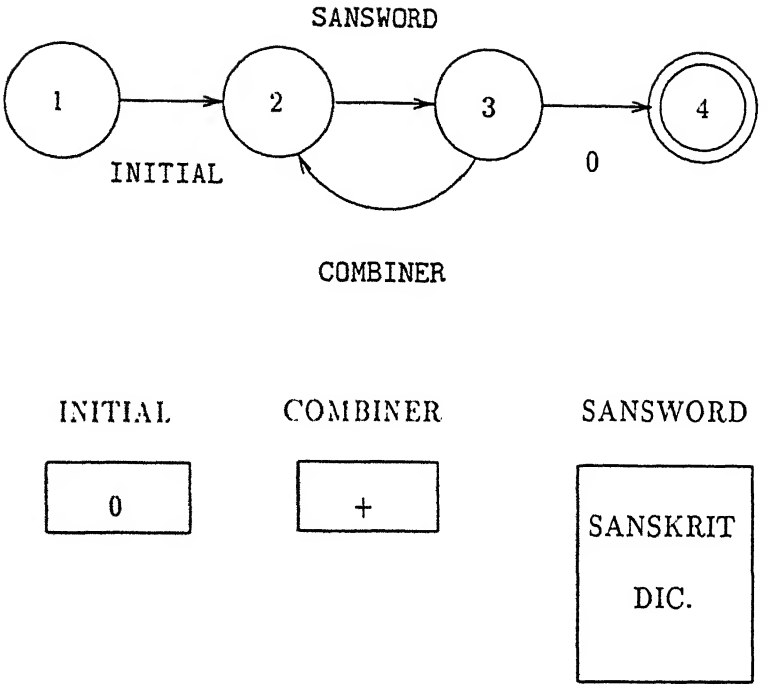


Figure 9: The Lexical Component

4.5 Implementation Details

KGEN was used to translate the two-level rules into FSTs. The use of KGEN necessitates using 0 as the NULL symbol and @ as the ANY symbol. KGEN requires that all correspondences (special and ordinary) be specified. This specifies the alphabet of the FSTs. The character ~ has been used to specify the BLANK character in the alphabet.

In Section 2.2 of Chapter 2 the varying notations in use for writing Sanskrit have been mentioned. The conventions used for this project has been:

1. The use of blanks for separating words is assumed. This is in accordance with the usual method followed.

2. The use of a single avagraha when an a is elided is also assumed. But when a A is elided no avagraha should be present. This is also in accordance with the usual method followed in book printed nowadays.

It should be noted that these conventions are not dictated by KIMMO. In fact by putting forth different two-level correspondences corresponding and then rewriting two-level rules and the alphabet accordingly any other desired notation might be used.

A lexicon of around 34,000 words has been used for the sublexicon SANSWORD. This includes words used in the Gita plus a dictionary of other prAwipaxikas. As of now the gloss entry in the lexicon is same as the lexical form itself.

While ideally speaking we need not deal with FSTs generated by KGEN, i.e. it should suffice that the rules be written in two-level form and there is no need to deal with PC-KIMMO except for the purposes of seeing results. In practice, it is found that the conversion of sandhi-rules is not perfect in the first go (similar to the writing of programs — it cannot be usually expected that a program works the first time it is successfully compiled). Because of this it is necessary to be familiar with how FSTs are generated from two-level notation. PC-KIMMO provides tracing facilities to debug the operation of the FSTs. Using these it is necessary to write two-level rules, test them out and if the results are incorrect to trace the operating of PC-KIMMO repeatedly.

4.6 Handling Irregular Cases

We now attempt to handle the 'irregular' cases discussed in the Chapter 2. It is here that we run into problems because the sandhi rules depend on the morphology of the words involed in sandhi. A few of these cases are described below:

Handling Sivehi As noted in Chapter 2 the sandhi of Siva + ehi is not Sivehi (as predicted by tables in Chapter 2). The correct form is Sivehi. This is due to the rule that a should combine with a succeeding e to form e if the e is of a verb form with a prefix A. The verb form ehi gets formed by adding the prefix A to ihi, hence the correct form will be Sivehi and not SivEhi.

One way to handle this 'irregularity' can be to use an external morphological analyzer e.g. *desika* (described in [17]. If over-generation and over-recognition are done i.e. both the forms SivAyom and SivAyOm are recognized and generation (by modifying the two-level rules) then (supposedly) the morphological analyzer can provide the morphological analysis and the relevant cases can be checked against and the correct form(s) found out.

But this runs into problems as there is no way to communicate to the morphological analyzer which rule has been applied so that the correct rule (and its corresponding form) can be found out. One remedy to this can be that all sandhi-rules be available to the morphological analyzer also. Thus all pattern-matching that is being done in sandhi-analyzer would have to be done here also.

Handling pragrihya cases As discussed in Chapter 2, Panini terms certain words to be pragrihya (pragqhya) and says that vowel sandhi should not be done if it is succeeded by a vowel. As an example the dual forms of words ending in I are pragrihya. Therefore, the dual form of the word hari — harI is pragrihya. Here again we run into similar problems discussed above. It is not possible to rule out the recognition or generation of a word (i.e. it is not possible to inhibit the application of some rule within the present formalism of PC-KIMMO) from the information available from the morphological analyzer.

Handling irregular visarga sandhi As noted earlier, visarga sandhi in Paninian derivational system is nothing but changes in final r of a word. For the indeclinables ending in r (e.g. punar) the sandhi rule gets modified. This case can be handled by using a special character (say α) for r in the lexicon. The gloss entry in the lexicon may contain r. While writing the two-level rule

here the rule should involve the special character α rather than r or visarga. Using this device, this particular case can be handled.

4.7 Summary

The use of KIMMO for sandhi analysis is now summarized. The rules component is built up using the following steps:

1. Write the two-level correspondence desired for each of the sandhi rules.
2. For each of the special character correspondences in the two-level correspondence write both the surface coercion rule and also the context restriction rule.
3. Iron out inconsistencies by grouping all the contexts of each of the special correspondences into a disjunction.

The two-level rules using above method is not unique but depends on the two-level correspondences used.

Chapter 5

Concluding Remarks

5.1 Summary of the Work Done

A sandhi-analyser for Sanskrit using PC-KIMMO has been built. In order to use PC-KIMMO, each of the sandhi-rules in Tables 1, 2 and 3 in Chapter 2 were converted into two-level correspondences. From these two-level rules were written down (the conversion process has been described in Chapter 4). A sample of these rules in the format accepted by KGEN is in Appendix B. From the character correspondences from these two-level correspondences the allowed set of pairs is found out. These rules were written down in the format accepted by KGEN which converted them into the format accepted by PC-KIMMO. (A sample of this format is given in Appendix C. The lexical component for PC-KIMMO is nothing but a finite state machine which ensures the order of words in a Sanskrit sentence. A sample of this lexical component (in the format acceptable to KIMMO) is given in Appendix D.

Chapter 5

Concluding Remarks

5.1 Summary of the Work Done

A sandhi-analyser for Sanskrit using PC-KIMMO has been built. In order to use PC-KIMMO, each of the sandhi-rules in Tables 1, 2 and 3 in Chapter 2 were converted into two-level correspondences. From these two-level rules were written down (the conversion process has been described in Chapter 4). A sample of these rules in the format accepted by KGEN is in Appendix B. From the character correspondences from these two-level correspondences the allowed set of pairs is found out. These rules were written down in the format accepted by KGEN which converted them into the format accepted by PC-KIMMO. (A sample of this format is given in Appendix C. The lexical component for PC-KIMMO is nothing but a finite state machine which ensures the order of words in a Sanskrit sentence. A sample of this lexical component (in the format acceptable to KIMMO) is given in Appendix D.

5.2 Evaluating KIMMO for Sandhi-Analysis

KIMMO is able to analyze sandhi for the sandhi rules mentioned in a tabular form (As in Tables 1,2 and 3 in Chapter 2). Though, these tables cover a vast majority of cases in Sandhi in Sanskrit, they nonetheless are not able to handle the 'irregularities' mentioned in Chapter 2. These 'irregularities', essentially, are the cases where sandhi rules are modified depending upon the morphology of the Sanskrit word. The question, whether it is possible to compile Paninian rules (which are able to handle all cases) directly into FSTs remains open.

The recognition phase in KIMMO (which is used for analysis of sandhi) operates as follows: The phonological rules (in the form of FSTs) try breaking up sandhis from left to right. The sequence of characters thus generated is compared against the finite state machine which checks whether the words thus generated are all in the lexicon or not. This comparison is done as soon as each character is generated using tries. A valid analysis is produced if each of the words generated is found in the lexicon. While it is possible to give these generated words to an external morphological analyzer (i.e. one not integrated within KIMMO along with Sandhi analysis) there is no way to pass the lexical and surface tapes to it in order to rule out certain correspondences not allowed in the 'irregular' cases.

Perhaps, a better approach would be to build a sandhi-analyzer and morphological analyzer both integrated within KIMMO.

5.3 Future Work

As indicated above, a morphological analyser integrated with sandhi-analyzer can be built. In this thesis an attempt has only been made to solve the sandhi-analysis at the sentence level. For analysing samasa (compounds) modifications can be done. Compound formation in Sanskrit, many a times, causes further changes in the words

participating in its formation besides normal sandhi rules. These will have to be handled to be able to analyze compounds.

Bibliography

[1] Antworth, Evan L. (1990) *PC-KIMMO: A Two-level Processor for Morphological Analysis.* Occasional Publications in Academic Computing, No. 16, Summer Institute of Linguistics.

[2] Antworth, Evan L. (1991) *ENGLEX: an English Lexicon for PC-KIMMO, ver. 1.0* Documentation accompanying PC-KIMMO ver. 1.0.

[3] Antworth, Evan L. (1994) *Morphological Parsing with a Unification-based Word Grammar.* North Texas Natural Language Processing Workshop, University of Texas at Arlington.

[4] Barton, Edward G. (1986) *Computational Complexity in Two-Level Morphology* 24th Annual Meeting of the ACL.

[5] Barton, Edward G., Robert C. Berwick and Eric Sven Ristad. (1987) *Computational Complexity and Natural Language.* MIT Press.

[6] Bhattacharya, Pushpak. (1986) *Computer Aid for Understanding Sanskrit: A First Step.* M.Tech. thesis, Department of Computer Science and Engineering, IIT Kanpur.

[7] Brandon, Frank Roberts. (1991) *Fonol: Phonological Programming Language.* Documentation accompanying the software FONOL ver 4.2.1.

[8] Bear, John. (1986) *A Morphological Recognizer with Syntactic and Phonological Rules* COLING-86.

[9] Bucknell, Roderick S. (1994) *Sanskrit Manual.* Motilal Banarsidass.

[10] Bharati, Akshar, Vineet Chaitanya and Rajeev Sangal (1995) *Natural Language Processing: A Paninian Perspective* Prentice Hall of India.

[11] Hopcraft, John E. and Jeffrey D. Ullman. (1979) *Introduction to Automata Theory, Languages and Computation* Addison-Wesley Publishing Co.

[12] Kaplan, Ronald M. and Kay, Martin. (1994) *Regular models of Phonological Rule Systems* Computational Linguistics, Vol. 20, No. 3.

[13] Koskenniemi Kimmo. (1984) *A general computational model for word-form recognition and production.* Proceedings of Coling-84, Association for Computational Linguistics.

[14] Koskenniemi, Kimmo and Church, Kenneth Ward (1988) *Complexity, Two-Level Morphology and Finnish.* Proceedings of Coling-88, Association for Computational Linguistics.

[15] Miles, Nathan. (1991) *KGEN User Manual.* Documentation accompanying PC-KIMMO ver. 1.0.

[16] Ramakrishnamacharyulu. Personal Communication.

[17] Ramanujan, P. (1992) *Computer Processing of Sanskrit.* 2nd conference on Computer Processing of Asian Languages, IIT Kanpur, (edited by R.M.K. Sinha).

[18] Ritchie, Graeme (1992) *Languages Generated by Two-Level Morphological Rules* Computational Linguistics, Vol. 18, No. 1.

[19] Shastri, Chakradhar Nautiyal 'Hans'. (1962) *Brihadanuvadachandrika.* Motilal Banarsidass.

[20] Shastri, Charudeva. (1973) *Vyakrana Chandrodaya - Vol. 5.* Motilal Banarsidass.

[21] Simpson, Candace J. and Narayanan Ramasamy. (1980) *paniniyanihshreniha.* Sandeepany West, Piercy, California.

[22] Sproat, Richard. (1992) *Morphology and Computation.* MIT Press.

[23] Vasu, S.C. (1962) *The Siddhantakaumudi of Bhattoji Dikshita — Vol. I.* Motilal Banarsidass.

[24] Whitney, D. W. (1924) *Sanskrit Grammar.* Motilal Banarsidass.

Appendix A

English to Devanagari Character Mapping

The under given English to Devanagari mapping has been used both within this thesis. This is a slightly modified form given in [10].

a	अ	A	आ	i	इ	I	ई	u	उ
U	ऊ	e	ए	E	ऐ	o	ओ	O	औ
L	ऋ	q	ऌ	Q	ॠ	H	ः	M	ं
z	ऽ								

k	क	K	ख	g	ग	G	घ	f	ङ
c	च	C	छ	j	ज	J	झ	F	ञ
t	त	T	थ	d	द	D	ध	N	न
w	ट	W	ठ	x	ड	X	ढ	n	ण
p	प	P	फ	b	ब	B	भ	m	म
y	य	r	र	l	ल	v	व	S	श
R	ष	s	स	h	ह				

Appendix B

Sample Rules File

A fragment of a sample rule file for KGEN is given below. KGEN converts the two-level rules listed in this file into the format accepted by PC-KIMMO. KGEN requires that a single rule be placed on a single line. But here long lines are shown split up by placing a \\ at its end.

```
; ';' is the comment character
; A line after '!' is copied directly onto the output
!;NULL 0
!;ANY  @
!;BOUNDARY #
!; avagraha Z
!; z = chandrabindu (e.g. Lz)
!; Q = deergha q
SUBSET Blank ^

!; Ordinary correspondences
```

```
PAIRS  a A i I u U e E o M q Q z L H Z
       a A i I u U e E o M q Q z L H Z
PAIRS  k K g G f c C j J F t T d D N w W x X n p P b B m
       k K g G f c C j J F t T d D N w W x X n p P b B m
PAIRS  y r l v S R s h
       y r l v S R s h

!; Special correspondences
PAIRS +  +
      0  ^
PAIRS k k t t h h h h   S  w w w w w w w p p
      g f d N G D X B   C  c t x j d l n b m
PAIRS m n n n n H H H H H H
      M F M N l s S R o 0 r
PAIRS a a a a a  a a A  A A A A    q q q o  o o 0 0
      0 Z A o 0  e E 0  E e o 0    r Q 0 0  0 a A 0
PAIRS i i  I u  u u U U  0 E  E e e e
      I 0  0 U  0 v v 0  v A  0 0 E a
PAIRS 0 0 0 0 0 0 0 0
      f s R S Z n v c

!; here follow the rules
RULE  k:g =>  ___ +:0  \\
          [r|g|G|x|X|b|B|y|v|j|J|d|D|l|a|A|i|I|u|U|o|O|e|E|q|Q|L]
RULE  k:f       =>   [a|A|i|I|u|U|o|O|e|E|q|Q|L]  ___ +:0 [n|m]
RULE  t:d =>  ___ +:0   \\
          [r|g|G|x|X|b|B|y|v|j|J|d|D|l|a|A|i|I|u|U|o|O|e|E|q|Q|L]
RULE  t:N =>  ___ +:0 [n|m]
RULE  h:G =>  k:k +:0 ___
RULE  h:D =>  t:t +:0 ___
RULE  h:X =>  w:w +:0 ___
```

```
RULE  h:B =>  p:p +:0 ___
RULE  h:B =>  p:p +:0 ___
RULE  w:t =>  ___ +:0 [T|t]
RULE  w:x =>  ___ +:0  \\
          [r|g|G|x|X|b|B|y|v|a|A|i|I|u|U|o|O|e|E|q|Q|L]
RULE  w:j =>  ___ +:0 [j|J]
RULE  w:d =>  ___ +:0 [d|D]
RULE  w:l =>  ___ +:0 l
RULE  w:n =>  ___ +:0 [n|m]
RULE  p:b =>  ___ +:0
          [r|g|G|x|X|b|B|y|v|j|J|d|D|l|a|A|i|I|u|U|o|O|e|E|q|Q|L]
RULE  p:m => ___ +:0 [n|m]

RULE  k:g <=  ___ +:0  \\
          [r|g|G|x|X|b|B|y|v|j|J|d|D|l|a|A|i|I|u|U|o|O|e|E|q|Q|L]
RULE  k:f       <=   [a|A|i|I|u|U|o|O|e|E|q|Q|L]  ___ +:0 [n|m]
RULE  t:d <=  ___ +:0   \\
          [r|g|G|x|X|b|B|y|v|j|J|d|D|l|a|A|i|I|u|U|o|O|e|E|q|Q|L]
RULE  t:N <=  ___ +:0 [n|m]
```

Appendix C

Sample Input File for PC-KIMMO

A fragment of sample file compiled by KGEN is given below. This sample file illustrates the format of rule file accepted by PC-KIMMO.

```
;NULL 0
;ANY  @
;BOUNDARY #
; avagraha Z
; z = chandrabindu (e.g. Lz)
; Q = deergha q
; Ordinary correspondences
; Special correspondences
; here follow the rules
ALPHABET
    a A i I u U e E o M q Q z L H Z k K g G f c C j J F t T d D N
w W x X n p P b B m y r l v S R s h + 0 ^
NULL 0
```

```
ANY @
BOUNDARY #
SUBSET Blank   ^

RULE "defaults" 1 31
    a A i I u U e E o M q Q z L H Z k K g G f c C j J F t T d D @
    a A i I u U e E o M q Q z L H Z k K g G f c C j J F t T d D @
 1: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

RULE "defaults" 1 31
    N w W x X n p P b B m y r l v S R s h + + k k t t h h h h S @
    N w W x X n p P b B m y r l v S R s h 0 ^ g f d N G D X B C @
 1: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

RULE "defaults" 1 31
    w w w w w w w p p m n n n n H H H H H H a a a a a a a A A A @
    c t x j d l n b m M F M N l s S R o 0 r 0 Z A o 0 e E 0 E e @
 1: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

RULE "defaults" 1 31
    A A q q q o o o 0 0 i i I u u u U U 0 E E e e e 0 0 0 0 0 0 @
    o 0 r Q 0 0 0 a A 0 I 0 0 U 0 v v 0 v A 0 0 E a f s R S Z n @
 1: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

RULE "defaults" 1 3
    0 0 @
    v c @
 1: 1 1 1

RULE "  t:N =>  ___ +:0 [n|m]" 3 5
      t  +  n  m  @
```

```
      N  0  n  m   @
 1:   2  1  1  1   1
 2.   0  3  0  0   0
 3.   0  0  1  1   0

RULE "  w:1 =>  ___ +:0 1" 3 4
      w  +  1   @
      1  0  1   @
 1:   2  1  1   1
 2.   0  3  0   0
 3.   0  0  1   0

RULE "  w:n =>  ___ +:0 [n|m]" 3 5
      w  +  n  m   @
      n  0  n  m   @
 1:   2  1  1  1   1
 2.   0  3  0  0   0
 3.   0  0  1  1   0
END
```

Appendix D

Sample Lexical Component File for PC-KIMMO

A sample lexical component file for PC-KIMMO follows.

```
ALTERNATION Begin SANSKRIT_WORD
ALTERNATION GotAWord COMBINER End

LEXICON INITIAL
0 Begin "[ "
LEXICON COMBINER
+ Begin " + "
LEXICON SANSKRIT_WORD
ABA GotAWord "ABA"
ABANaka GotAWord "ABANaka"
ABAsa GotAWord "ABAsa"
ABIkRNya GotAWord "ABIkRNya"
ABIla GotAWord "ABIla"
yuxXe GotAWord "yuxXe"
```

```
yuyuXAnaH GotAWord "yuyuXAnaH"
yuyuwsavaH GotAWord "yuyuwsavaH"
yuyuwsum GotAWord "yuyuwsum"
LEXICON End
0 # " ]"
END
```

Appendix E

Sample Run of PC-KIMMO

```
$ pckimmo
PC-KIMMO TWO-LEVEL PROCESSOR
Version 1.0.8 (18 February, 1992), Copyright 1992 SIL
Type ? for help
PC-KIMMO>LOAD RULES sandhi
Rules being loaded from sandhi.rul
PC-KIMMO>LOAD LEXICON sandhi
Lexicon being loaded from sandhi.lex
PC-KIMMO>generate rAma+avawAra
  rAmAvawAra

PC-KIMMO>generate naxi+ISa
  naxISa

PC-KIMMO>recognize naxISa
  naxI+ISa      [ naxI + ISa ]

PC-KIMMO>recognize rAmAvawAra
```

```
  rAmA+avawAra      [ rAmA + avawAra ]
  rAma+avawAra      [ rAma + avawAra ]
PC-KIMMO>recognize SivAlayeSa
  SivA+Alaya+ISa      [ SivA + Alaya + ISa ]
  SivAlaya+ISa      [ SivAlaya + ISa ]
  Siva+Alaya+ISa      [ Siva + Alaya + ISa ]
PC-KIMMO>recognize namaswe
  namaH+we      [ namaH + we ]

PC-KIMMO>recognize xAsoZham
  *** NONE ***

PC-KIMMO>recognize soZham
  saH+aham      [ saH + aham ]

PC-KIMMO>recognize prawyAhAra
  prawyAhAra      [ prawyAhAra ]
  prawi+AhAra      [ prawi + AhAra ]
  prawi+Aha+ara      [ prawi + Aha + ara ]
  prawi+Aha+Ara      [ prawi + Aha + Ara ]

PC-KIMMO>recognize rAmo^gacCawi
  rAmaH+gacCawi      [ rAmaH + gacCawi ]
```